In [1]:
```
from vatic.engines import Simulator
from vatic.data.loaders import load_input, RtsLoader
from vatic.engines import Simulator

import pandas as pd
import numpy as np

from pathlib import Path
import dill as pickle
import os
from datetime import datetime
import matplotlib.pyplot as plt
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
RUC_MIPGAPS = {'Texas-7k': 0.01}
SCED_HORIZONS = {'Texas-7k': 4}

grid = 'Texas-7k'
num_days = 1
start_date = '2018-01-19' #for Texas pick a date in 2018
init_state_file = None
template, gen_data, load_data = load_input(grid, start_date,
    num_days=num_days, init_state_file=init_state_file)
```

In [ ]:
```
siml = Simulator(template, gen_data, load_data, None,
                 pd.to_datetime(start_date).date(), 1, solver='gurobi',
                 solver_options={"Threads": 8}, run_lmps=False, mipgap=RUC_MI
                 load_shed_penalty = 1e4, reserve_shortfall_penalty = 1e3,
                 reserve_factor=0.05, output_detail=3,
                 prescient_sced_forecasts=True, ruc_prescience_hour=0,
                 ruc_execution_hour=16, ruc_every_hours=24,
                 ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
                 lmp_shortfall_costs=False,
                 enforce_sced_shutdown_ramprate=False,
                 no_startup_shutdown_curves=False,
                 init_ruc_file=None, verbosity=0,
                 output_max_decimals=4, create_plots=False,
                 renew_costs=None, save_to_csv=False,
                 last_conditions_file=None,)
report_dfs = siml.simulate()
```

In [5]:
```
cols_gen_data = list(gen_data.columns)
```

In [ ]:
```

```

In [ ]:

Part A

(i)

Run a series of simulations by scaling renewable generation using the following factors:

$$\alpha = 1.0, 0.75, 0.5, 0.25, 0.0.$$

That is, multiply all renewable generation (wind and solar) by each of these scaling parameters and rerun the `Simulator` for each case.
Then, **plot the results** (for example, total system cost, market price and thermal dispatch) against α.

Finally, **comment on the observed results**, discuss how reducing renewable generation impacts total cost and prices.

(ii)

Examine whether there is a **linear relationship** between renewable generation and the key output variables (e.g., total cost, prices, and thermal dispatch).

In particular, does the change in each output appear **proportional** to the change in α, or do you observe **nonlinear effects** (such as thresholds, curvatures, or sharp transitions in the plots)?

This analysis can be done by taking the **average value throughout the day** for each output variable and comparing these averages across different values of α.

In [4]:
```
import re
import numpy as np
import pandas as pd

RENEW_PAT = re.compile(r'(OnshoreWindTurbine|SolarPhotovoltaic|RTPV)', re.IG

def get_actl_renewable_cols(gen_data):
    """
    Return a list of MultiIndex column labels (level0='actl') that
    correspond to renewable generators (wind/solar/RTPV).
    """
    assert isinstance(gen_data.columns, pd.MultiIndex), \
        "gen_data must have MultiIndex columns (level0: actl/fcst/...)"
```

```
    ren_cols = []
    for lvl0, name in gen_data.columns:
        if lvl0 == 'actl' and RENEW_PAT.search(str(name)):
            ren_cols.append((lvl0, name))

    print(f"Found {len(ren_cols)} actl renewable columns.")
    return ren_cols

def scale_renewables_actl(gen_data, alpha, ren_cols=None):
    """
    Return a copy of gen_data where only the actl renewable columns
    have been multiplied by alpha.
    """
    if ren_cols is None:
        ren_cols = get_actl_renewable_cols(gen_data)

    g2 = gen_data.copy()

    for col in ren_cols:
        g2[col] = g2[col] * alpha

    return g2
```

In [5]:
```
import pickle

alphas = [1.0, 0.75, 0.5, 0.25, 0.0]

actl_ren_cols = get_actl_renewable_cols(gen_data)
```

```
Found 185 actl renewable columns.
```

In [ ]:
```
reports_by_alpha = {}

for a in alphas:
    print(f"\nRunning simulation with α = {a:.2f}")

    g_scaled = scale_renewables_actl(gen_data, a, ren_cols=actl_ren_cols)

    sim = Simulator(
        template, g_scaled, load_data, None,
        pd.to_datetime(start_date).date(), 1,
        solver='gurobi', solver_options={"Threads": 8},
        run_lmps=False, mipgap=RUC_MIPGAPS[grid],
        load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
        reserve_factor=0.05, output_detail=3,
        prescient_sced_forecasts=True, ruc_prescience_hour=0,
        ruc_execution_hour=16, ruc_every_hours=24,
        ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
        lmp_shortfall_costs=False,
        enforce_sced_shutdown_ramprate=False,
```

```
        no_startup_shutdown_curves=False,
        init_ruc_file=None, verbosity=0,
        output_max_decimals=4, create_plots=False,
        renew_costs=None, save_to_csv=False,
        last_conditions_file=None,
    )

    report_dfs = sim.simulate()
    reports_by_alpha[a] = report_dfs


    tag = f"{int(round(a * 100)):03d}"
    filename = f"report_by_alpha_{tag}.pkl"

    with open(filename, "wb") as f:
        pickle.dump(report_dfs, f)

    print(f"Saved results for α={a:.2f} to {filename}")
```

In [6]:
```
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

alpha_to_file = {
    1.00: "report_by_alpha_100.pkl",
    0.75: "report_by_alpha_075.pkl",
    0.50: "report_by_alpha_050.pkl",
    0.25: "report_by_alpha_025.pkl",
    0.00: "report_by_alpha_000.pkl",
}

reports_by_alpha = {}
for a, fname in alpha_to_file.items():
    with open(fname, "rb") as f:
        reports_by_alpha[a] = pickle.load(f)
```

In [ ]:
```
```

In [7]:
```
def prepare_hourly(rep):
    hs = rep["hourly_summary"].copy()
    hs = hs.reset_index()

    # timestamp
    hs["Datetime"] = pd.to_datetime(hs["Date"]) + pd.to_timedelta(hs["Hour"]

    # total cost per hour
    hs["TotalCost"] = hs["FixedCosts"] + hs["VariableCosts"]
```

```
    # Curtailment as percentage of available renewables
    hs["CurtailmentPct"] = (
        hs["RenewablesCurtailment"] / hs["RenewablesAvailable"]
    ) * 100.0
    hs["CurtailmentPct"] = hs["CurtailmentPct"].replace([np.inf, -np.inf], r

    hs = hs.set_index("Datetime").sort_index()
    return hs

hourly_by_alpha = {a: prepare_hourly(rep) for a, rep in reports_by_alpha.ite
```

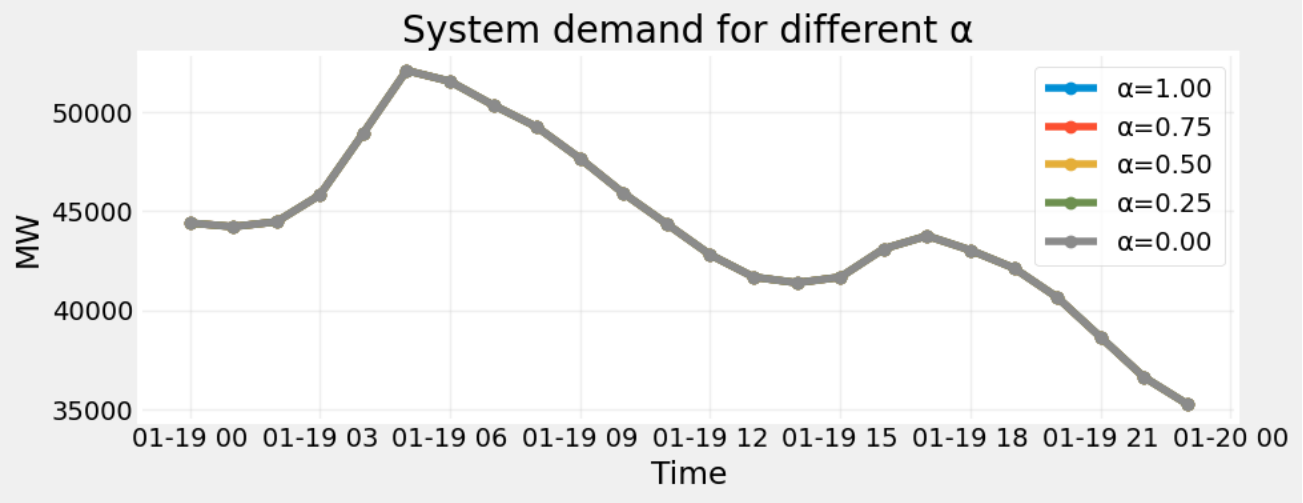
In [ ]:

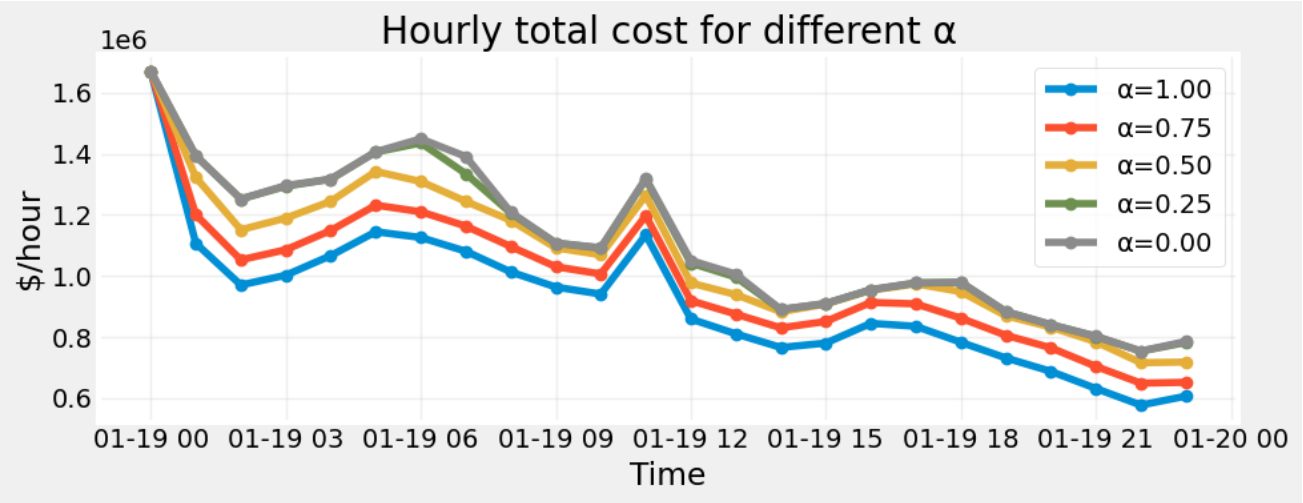
In [8]:
```
def summarize_alpha(hs):
    total_cost_day = hs["TotalCost"].sum()

    return pd.Series(
        {
            "total_cost_day": total_cost_day,
        }
    )

metrics_by_alpha = {}
for a, hs in hourly_by_alpha.items():
    metrics_by_alpha[a] = summarize_alpha(hs)

metrics_df = pd.DataFrame(metrics_by_alpha).T.sort_index()
display(metrics_df)
```

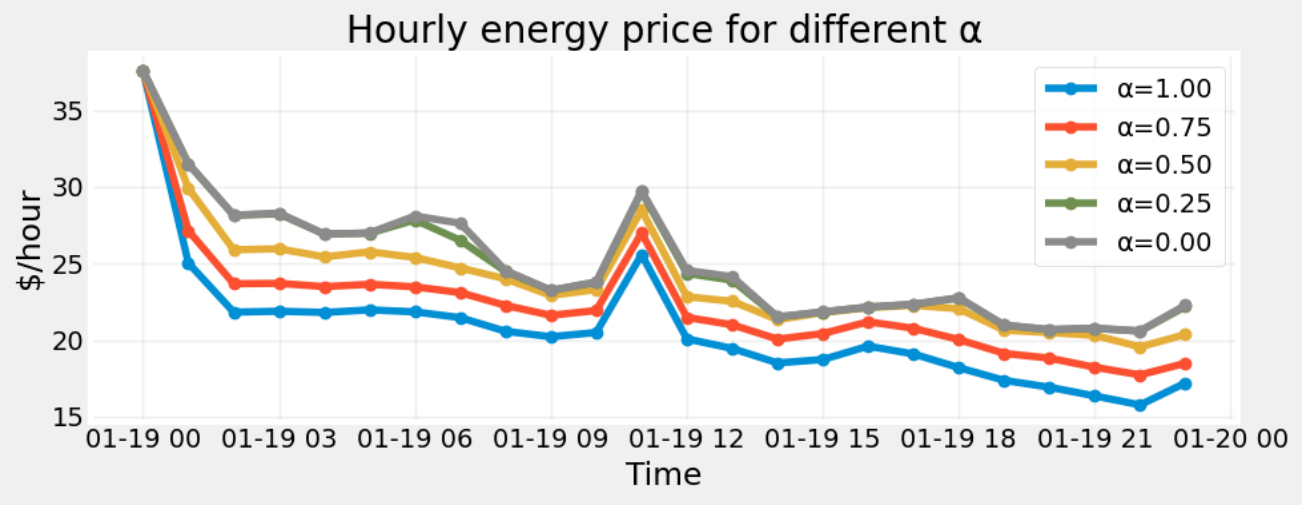
	total_cost_day
0.00	2.673682e+07
0.25	2.663218e+07
0.50	2.558317e+07
0.75	2.382893e+07
1.00	2.213490e+07

In [9]:
```
def plot_metric(metric, ylabel, title, scenarios=hourly_by_alpha):
    plt.figure(figsize=(10, 4))
    for a, hs in sorted(scenarios.items(), reverse=True):
        if metric not in hs.columns:
            continue
        label = f"α={a:.2f}"
        plt.plot(hs.index, hs[metric], marker="o", label=label)
    plt.xlabel("Time")
    plt.ylabel(ylabel)
```

```
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()

plot_metric("Demand", "MW", "System demand for different α")
plot_metric("TotalCost", "$/hour", "Hourly total cost for different α")
plot_metric("Price", "$/hour", "Hourly energy price for different α")
```



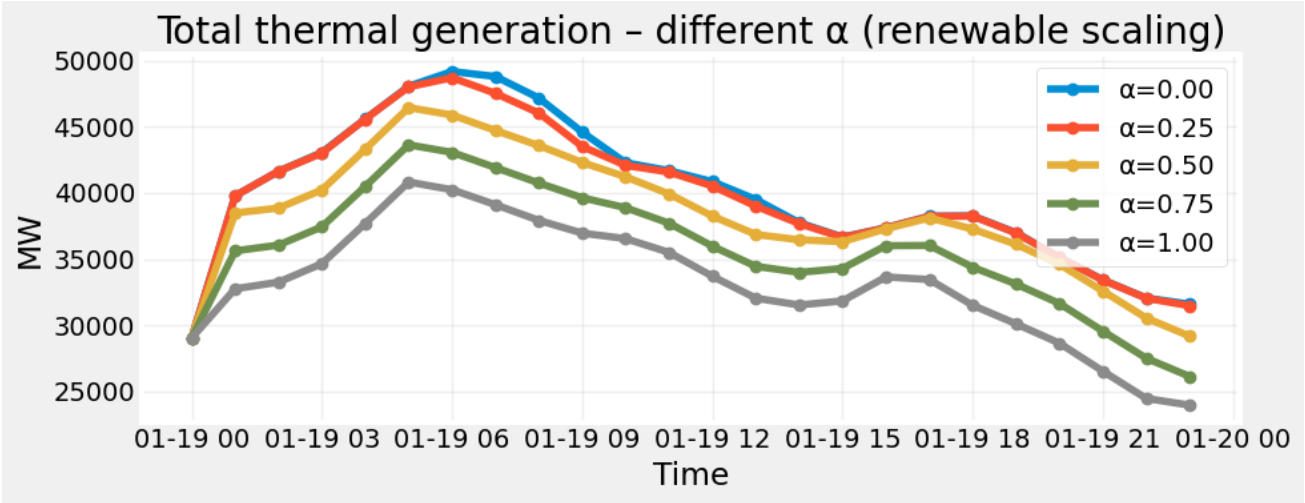


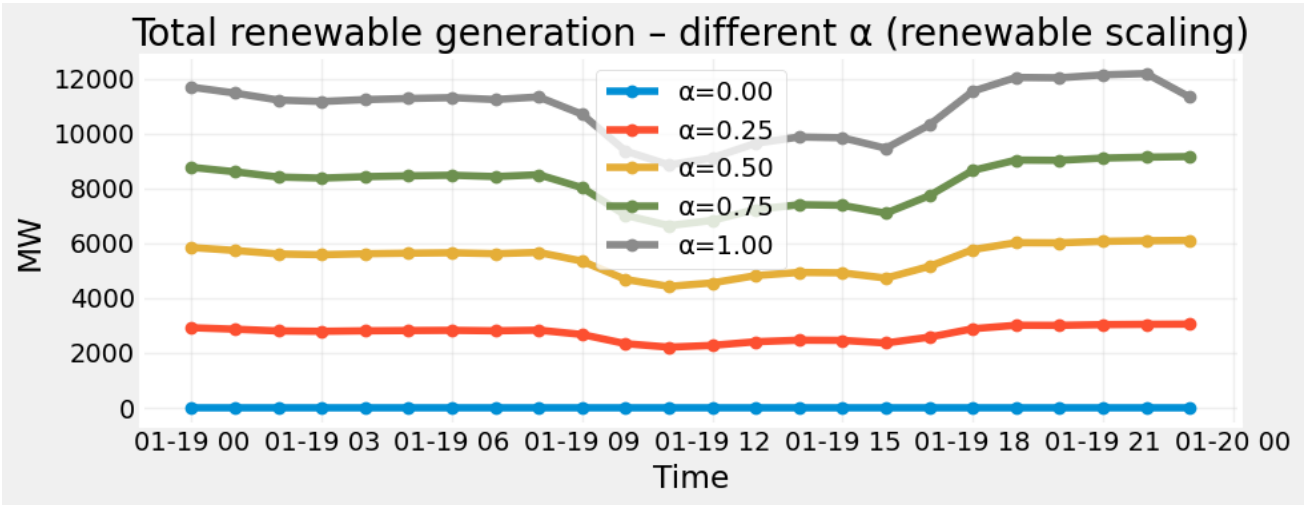


In [ ]:

In [10]:
```
scenarios_raw_alpha = {
    f"α={a:.2f}": rep for a, rep in sorted(reports_by_alpha.items())
}
def thermal_dispatch_time_series(rep):
    td = rep["thermal_detail"].copy()
    td = td.reset_index()
    td["Datetime"] = pd.to_datetime(td["Date"]) + pd.to_timedelta(td["Hour"]
    return td.groupby("Datetime")["Dispatch"].sum().sort_index()

def renewable_dispatch_time_series(rep):
    rd = rep["renew_detail"].copy()
    rd = rd.reset_index()
    rd["Datetime"] = pd.to_datetime(rd["Date"]) + pd.to_timedelta(rd["Hour"]
    return rd.groupby("Datetime")["Output"].sum().sort_index()
thermal_ts_alpha = {
    name: thermal_dispatch_time_series(rep)
    for name, rep in scenarios_raw_alpha.items()
}

renew_ts_alpha = {
    name: renewable_dispatch_time_series(rep)
    for name, rep in scenarios_raw_alpha.items()
}
# Thermal dispatch time series
plt.figure(figsize=(10, 4))
for name, series in thermal_ts_alpha.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total thermal generation — different α (renewable scaling)")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```

```
# Renewable dispatch time series
plt.figure(figsize=(10, 4))
for name, series in renew_ts_alpha.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total renewable generation – different α (renewable scaling)")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```





Comments for Part (a) (i):

- The system demand profile is identical for all values of α. All curves lie exactly on top of each other because demand is taken as an exogenous input to the model. Scaling renewable generation does not change the underlying load. The system still has to serve the same hourly demand regardless of how much wind/solar is

available.

- Total system cost is monotonically increasing as α decreases. For each hour, the α = 1.0 case (full renewables) yields the lowest production cost, and costs rise progressively for α = 0.75, 0.50, 0.25, and are highest at α = 0.0 (no renewables). This confirms the economic intuition that removing zero-marginal-cost renewables forces the system to rely on higher-cost thermal generation, pushing up total operating costs.

- The hourly energy price shows the same ordering as total cost where prices are lowest with α = 1.0 and increase as renewable penetration is reduced. In tight hours (around the load peak and the mid-day ramp), the price spread between α values widens, indicating that when the system is stressed, the loss of renewables shifts the marginal unit to more expensive generators and raises the clearing price more sharply. In lower-load hours, the price differences are smaller but still consistently ordered by α, reflecting the merit-order effect of renewables.

- Thermal generation moves in the opposite direction to renewables. For every hour, thermal output is highest when α = 0.0 (no renewables) and decreases steadily as α increases to 1.0. The shape of the curve across the day follows the load profile, but the vertical level depends on α: with higher renewable penetration, thermal plants are displaced and run less, especially during mid-day hours when wind/solar output is relatively strong. This shows that renewables and thermal units largely substitute for each other in meeting demand.

- Renewable generation scales almost proportionally with α. The α = 1.0 curve shows the baseline wind/solar profile. α = 0.75, 0.50, and 0.25 lie below it with similar shapes and approximately scaled magnitudes, and α = 0.0 corresponds to essentially zero renewable output. The fact that all renewable curves have the same temporal pattern but different vertical levels reflects the way we constructed the experiment which is where we multiply the baseline renewable profile by α without changing its hour-to-hour shape.

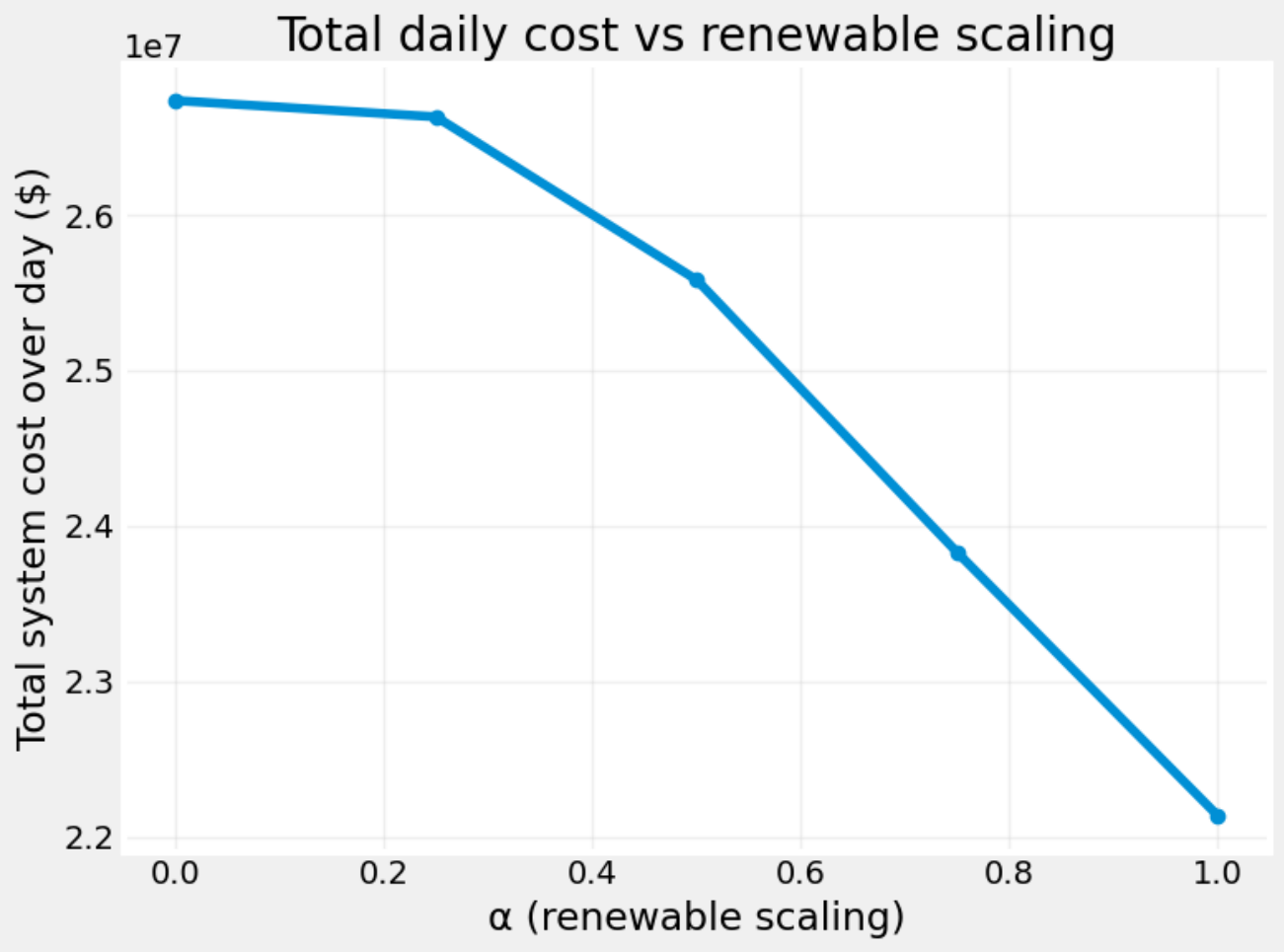
In [11]:
```
thermal_ts_by_alpha = {a: thermal_dispatch_time_series(rep)
                       for a, rep in reports_by_alpha.items()}

renew_ts_by_alpha   = {a: renewable_dispatch_time_series(rep)
                       for a, rep in reports_by_alpha.items()}
```

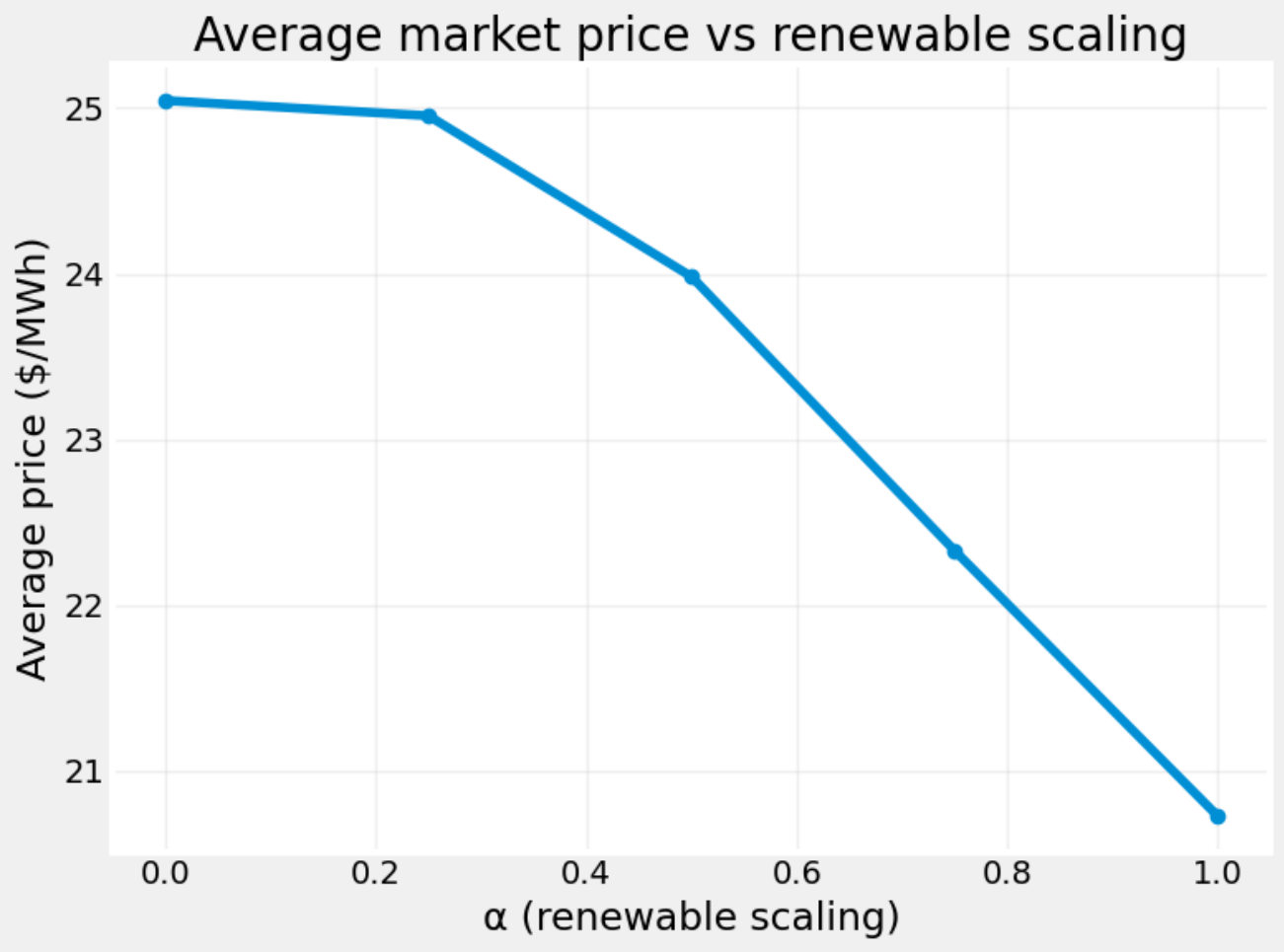
In [12]:
```
def summarize_alpha_full(alpha, hs):
    """
```

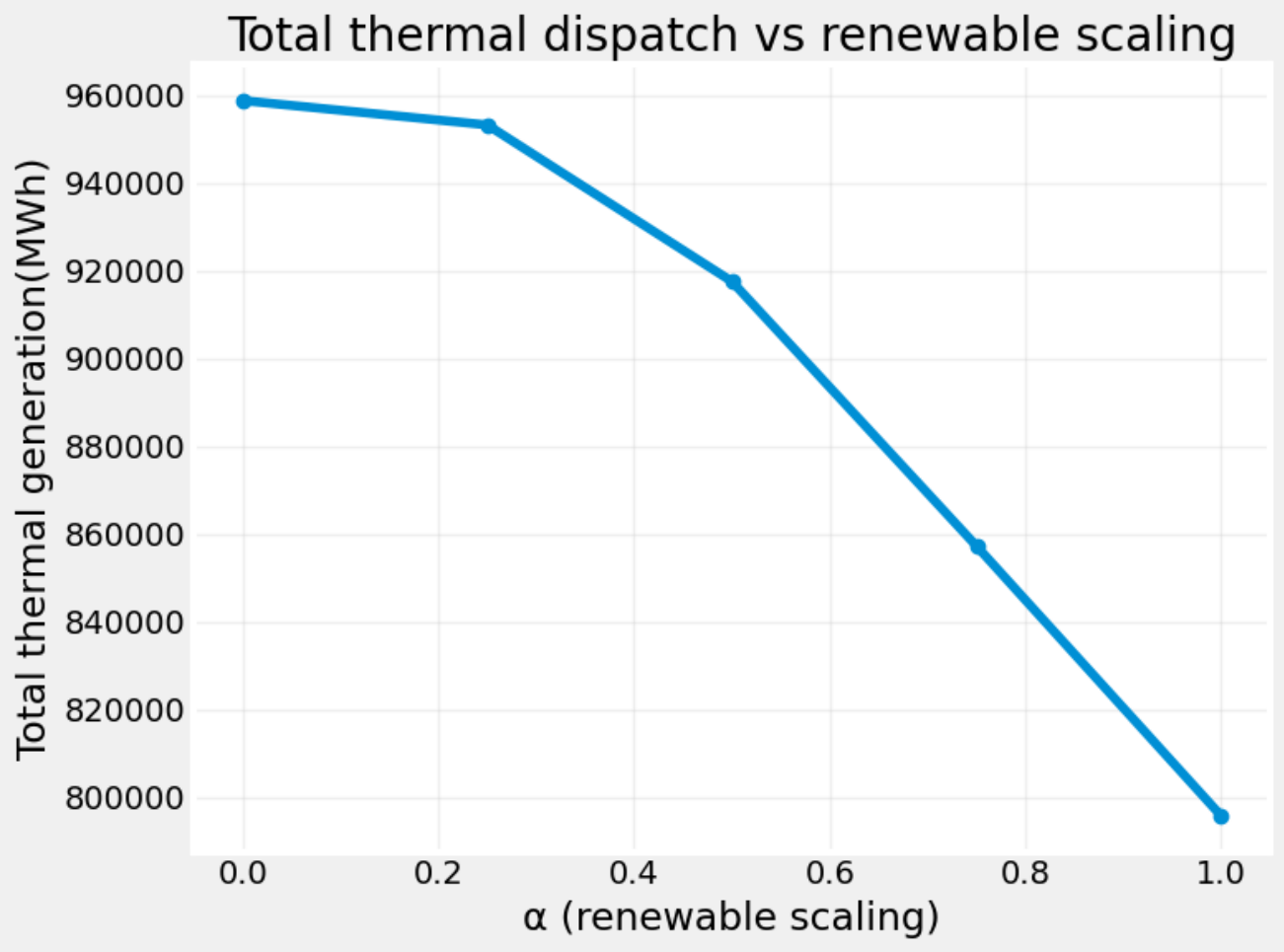
```
    alpha: scalar (0.0, 0.25, ...)
    hs:    hourly_by_alpha[alpha]
    Uses thermal_ts_by_alpha and renew_ts_by_alpha defined above.
    """
    # daily totals / averages for cost
    total_cost_day          = hs["TotalCost"].sum()
    avg_total_cost_per_hour = hs["TotalCost"].mean()

    # average energy price over the day
    avg_price = hs["Price"].mean()

    # thermal dispatch
    thermal_series = thermal_ts_by_alpha[alpha]
    total_thermal_MWh = thermal_series.sum()
    avg_thermal_MW    = thermal_series.mean()

    # renewable dispatch
    renew_series = renew_ts_by_alpha[alpha]
    total_renew_MWh = renew_series.sum()
    avg_renew_MW    = renew_series.mean()

    return pd.Series({
        "total_cost_day":          total_cost_day,
        "avg_total_cost_per_hour": avg_total_cost_per_hour,
        "avg_price":               avg_price,
        "total_thermal_MWh":       total_thermal_MWh,
        "avg_thermal_MW":          avg_thermal_MW,
        "total_renew_MWh":         total_renew_MWh,
        "avg_renew_MW":            avg_renew_MW,
    })

metrics_by_alpha = {}
for a in sorted(reports_by_alpha.keys()):
    hs = hourly_by_alpha[a]
    metrics_by_alpha[a] = summarize_alpha_full(a, hs)

metrics_df = pd.DataFrame(metrics_by_alpha).T.sort_index()
display(metrics_df)
```

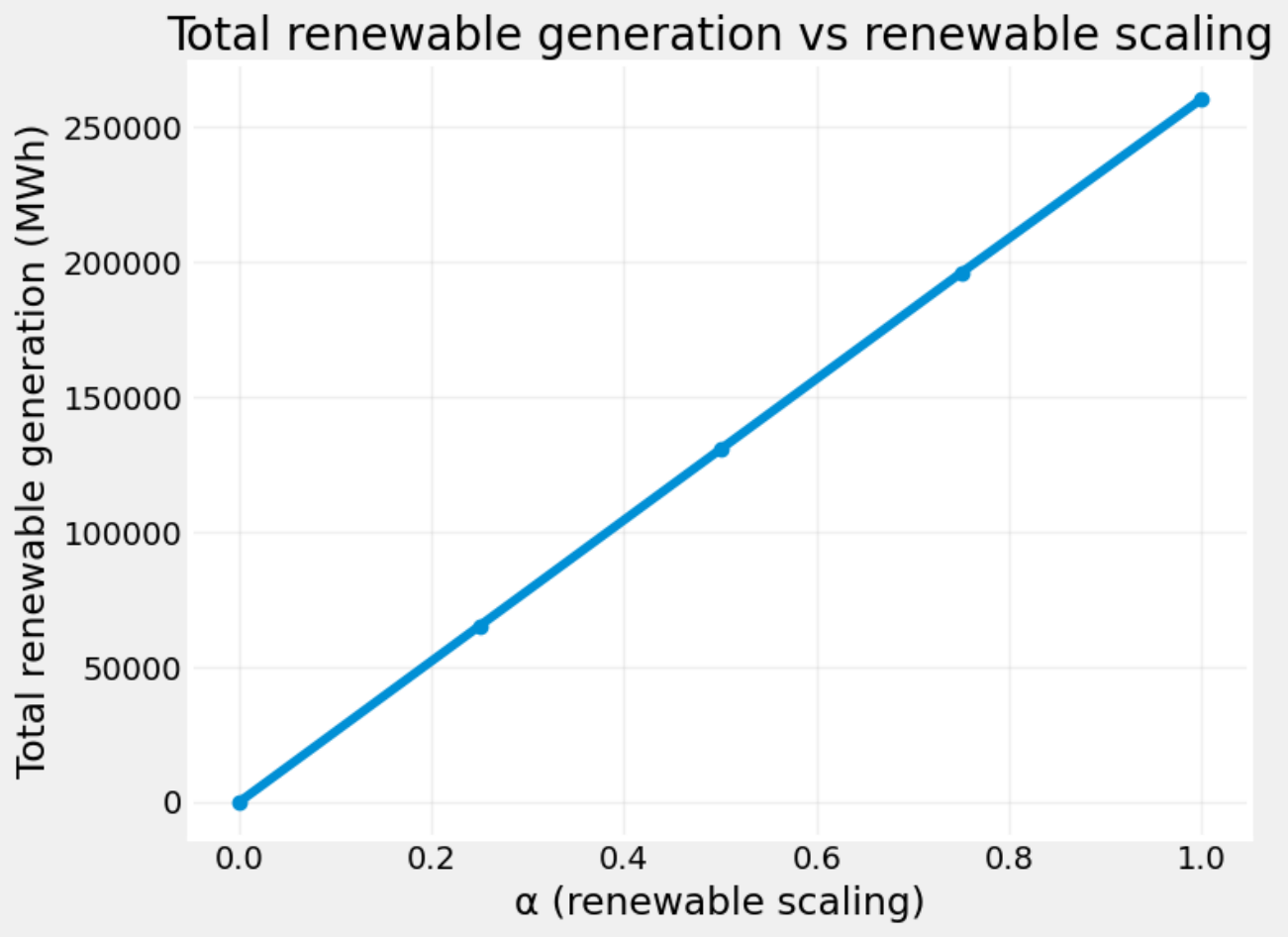
	total_cost_day	avg_total_cost_per_hour	avg_price	total_thermal_MWh	avg_the
0.00	2.673682e+07	1.114034e+06	25.044471	958817.726310	3995
0.25	2.663218e+07	1.109674e+06	24.952846	953262.785666	397
0.50	2.558317e+07	1.065965e+06	23.982412	917578.773767	3823
0.75	2.382893e+07	9.928721e+05	22.330658	857248.920819	357
1.00	2.213490e+07	9.222877e+05	20.730421	795750.426047	331

In [13]:
```
alphas = metrics_df.index.values

# Cost vs alpha
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["total_cost_day"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Total system cost over day ($)")
plt.title("Total daily cost vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Average price vs alpha
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["avg_price"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Average price ($/MWh)")
plt.title("Average market price vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Thermal dispatch vs alpha
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["total_thermal_MWh"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Total thermal generation(MWh)")
plt.title("Total thermal dispatch vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()

# Average renewable generation vs alpha — would be linear by construction
plt.figure(figsize=(8,6))
plt.plot(alphas, metrics_df["total_renew_MWh"], marker="o")
plt.xlabel("α (renewable scaling)")
plt.ylabel("Total renewable generation (MWh)")
plt.title("Total renewable generation vs renewable scaling")
plt.grid(True, alpha=0.3)
plt.tight_layout()
```


Total daily cost vs renewable scaling
1e7
Total system cost over day ($)
2.6
2.5
2.4
2.3
2.2
0.0
0.2
0.4
0.6
0.8
1.0
α (renewable scaling)


Average market price vs renewable scaling
Average price ($/MWh)
25
24
23
22
21
0.0
0.2
0.4
0.6
0.8
1.0
α (renewable scaling)


Total thermal dispatch vs renewable scaling
Total thermal generation(MWh)
960000
940000
920000
900000
880000
860000
840000
820000
800000
0.0
0.2
0.4
0.6
0.8
1.0
α (renewable scaling)



In [14]:
```
def linfit_and_r2(x, y):
    coeffs = np.polyfit(x, y, 1)
    y_hat  = np.polyval(coeffs, x)
    ss_res = np.sum((y - y_hat)**2)
    ss_tot = np.sum((y - np.mean(y))**2)
    r2 = 1 - ss_res/ss_tot
    return coeffs, r2

for col in ["total_cost_day", "avg_price", "total_thermal_MWh"]:
    coeffs, r2 = linfit_and_r2(alphas, metrics_df[col].values)
    print(col, "linear fit:", coeffs, "R^2 =", round(r2, 4))
```

```
total_cost_day linear fit: [-4802829.84520002 27384615.35732   ] R^2 = 0.924
2
avg_price linear fit: [-4.500115   25.65821917] R^2 = 0.9222
total_thermal_MWh linear fit: [-168859.38614942  980961.41959652] R^2 = 0.92
62
```

Comments for Part (a) (i):

- The table before the plots shows that as α increases, renewable generation (total_renew_MWh) grows almost perfectly proportionally. Correspondingly, thermal generation, average price, and total system cost all decrease. The declines are smooth and monotonic, supporting an approximately linear inverse relationship between renewable availability and these outputs.

- Total system cost declines steadily as α increases. The slope of the cost curve is fairly constant and smooth, with no visible curvature or threshold. This indicates that total cost decreases approximately linearly with renewable generation. The linear regression confirms this, with an R^2 =0.92, indicating strong linearity.

- Average energy prices fall consistently as α increases. The price drop occurs in a mostly straight-line fashion without nonlinear jumps or sharp transitions. This is expected because more renewables displace higher-cost thermal units. The relationship is again close to linear, supported by a high regression R^2 =0.92.

- Thermal output decreases monotonically as renewables scale up. The curve has no visible inflection points. Instead, the decline is nearly uniform. This suggests a roughly linear substitution effect where each marginal increase in renewable availability offsets a proportional amount of thermal generation. The regression indicates strong linearity (R^2 =0.93).

- Renewable generation increases almost perfectly proportionally with α. The plot is nearly a straight line passing through the origin, confirming that the scaling procedure is linear by construction. This validates using α as an appropriate linear displacer for renewable availability.

- Regarding the regression, the fitted lines for cost, price, and thermal dispatch all have high R^2 values (0.92–0.93), showing strong linear relationships. The slopes are negative for cost, price, and thermal output, confirming that increasing renewable penetration reduces system cost, market prices, and thermal dispatch at nearly constant marginal rates.

Part B

Wind generation is inherently uncertain due to the variability of weather systems. To evaluate how fluctuations influences system operation and market outcomes, we introduce controlled perturbations to the baseline wind production data. We consider two cases. (Here wind turbines refer to all sources that have 'Wind' in their name)

(i) Independent Forecast Errors

In the first case, we assume that the variations for each wind turbine are independent across generators. That is, each turbine's production is adjusted by an error term drawn independently from the same statistical distribution.

We want to find the answer to "what if wind were 15% higher on average that day? ($\mu = 15$). The volatility around this average bias is $\sigma = 20$.

Each turbine's error is drawn independently of the others, $\epsilon_i(t) \sim \mathcal{N}(\mu, \sigma^2)$.

Mathematically, for turbine i at time t: $production_i(t) = baseline_i(t) \cdot (1 + \epsilon_i(t))$

(ii) Correlated Errors (Shared Weather)

Turbines experience the correlated/similar regional weather, so we model a shared shock plus a small local wiggle (volatility):

$$\epsilon_i(t) \;=\; \mu \;+\; \sigma\Big(\sqrt{\rho}\, z^{\text{common}}(t) \;+\; \sqrt{1-\rho}\, z_i^{\text{local}}(t)\Big)\,,$$

with $z^{\text{common}}(t), z_i^{\text{local}}(t) \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$, correlation $\rho = 0.8$, and the same $\mu = 0.15$, $\sigma = 0.20$ as above. Properties:

- $z^{\text{common}}(t)$ is the same across all sources at a given time t.
- At any hour t, turbines move mostly together (correlation ρ across units).
- Across time, errors remain independent (hourly weather surprises are independent draws).

In [ ]:

In [15]:
```
import numpy as np
import pandas as pd

# Parameters
MU    = 0.15
SIGMA = 0.20
RHO   = 0.8

def get_actl_wind_cols(gen_data):

    assert isinstance(gen_data.columns, pd.MultiIndex)
    wind_cols = []
    for lvl0, name in gen_data.columns:
        if lvl0 == "actl" and "wind" in str(name).lower():
```

```
            wind_cols.append((lvl0, name))
    print(f"Found {len(wind_cols)} actl wind columns.")
    return wind_cols
```

In [16]:
```
wind_cols = get_actl_wind_cols(gen_data)
wind_cols[:5]
```

```
Found 149 actl wind columns.
```

Out[16]:
```
[('actl', '54979_OnshoreWindTurbine_WIND'),
 ('actl', '55581_OnshoreWindTurbine_EXIS'),
 ('actl', '55747_OnshoreWindTurbine_NWP2'),
 ('actl', '55968_OnshoreWindTurbine_WTG1'),
 ('actl', '55992_OnshoreWindTurbine_1')]
```

In [17]:
```
def apply_wind_noise_independent(gen_data, mu=MU, sigma=SIGMA, seed=None):

    if seed is not None:
        np.random.seed(seed)

    wind_cols = get_actl_wind_cols(gen_data)
    g_ind = gen_data.copy()

    n_hours = len(gen_data.index)

    for col in wind_cols:
        # independent epsilon_i for this turbine over all hours
        eps = np.random.normal(loc=mu, scale=sigma, size=n_hours)
        # new production = baseline * (1 + eps)
        g_ind[col] = gen_data[col].values * (1.0 + eps)

    return g_ind
```

In [18]:
```
gen_data_ind = apply_wind_noise_independent(gen_data, seed=42)

# simulation
sim_ind = Simulator(
    template, gen_data_ind, load_data, None,
    pd.to_datetime(start_date).date(), 1,
    solver='gurobi', solver_options={"Threads": 8},
    run_lmps=False, mipgap=RUC_MIPGAPS[grid],
    load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
    reserve_factor=0.05, output_detail=3,
    prescient_sced_forecasts=True, ruc_prescience_hour=0,
    ruc_execution_hour=16, ruc_every_hours=24,
    ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
    lmp_shortfall_costs=False,
    enforce_sced_shutdown_ramprate=False,
    no_startup_shutdown_curves=False,
    init_ruc_file=None, verbosity=0,
```

```
    output_max_decimals=4, create_plots=False,
    renew_costs=None, save_to_csv=False,
    last_conditions_file=None,
)
report_ind = sim_ind.simulate()
```

```
Found 149 actl wind columns.
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-packages/vatic/models/params.py:1022)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:126)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:127)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:130)
Calculating PTDF Matrix Factorization
WARNING: DEPRECATED: The quicksum(linear=...) argument is deprecated and
ignored.  (deprecated in 6.6.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/uc_utils.py:100)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/power_vars.py:63)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/power_vars.py:66)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/power_vars.py:58)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
```

```
packages/egret/model_library/unit_commitment/power_vars.py:54)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:198)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:199)
WARNING: DEPRECATED: Using __getitem__ to return a set value from its
(ordered) position is deprecated.  Please use at()  (deprecated in 6.1, will
be removed in (or after) 7.0) (called from /home/rp7104/.conda/envs/vatic-
test/lib/python3.11/site-
packages/egret/model_library/unit_commitment/startup_costs.py:239)
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
```

In [ ]:

In [19]:
```python
def apply_wind_noise_correlated(gen_data, mu=MU, sigma=SIGMA, rho=RHO, seed=

    if seed is not None:
        np.random.seed(seed)

    wind_cols = get_actl_wind_cols(gen_data)
    g_corr = gen_data.copy()

    times = gen_data.index
    T = len(times)
    Nw = len(wind_cols)

    # shared shock (one per hour)
    z_common = np.random.normal(loc=0.0, scale=1.0, size=T)

    # local wiggles (per hour per turbine)
    z_local  = np.random.normal(loc=0.0, scale=1.0, size=(T, Nw))

    # loop over hours and turbines
```

```
    for t_idx, t in enumerate(times):
        for j, col in enumerate(wind_cols):
            eps_ij = mu + sigma * (
                np.sqrt(rho)     * z_common[t_idx] +
                np.sqrt(1 - rho) * z_local[t_idx, j]
            )
            baseline = gen_data.at[t, col]
            g_corr.at[t, col] = baseline * (1.0 + eps_ij)

    return g_corr
```

In [20]:
```
gen_data_corr = apply_wind_noise_correlated(gen_data, seed=42)

sim_corr = Simulator(
    template, gen_data_corr, load_data, None,
    pd.to_datetime(start_date).date(), 1,
    solver='gurobi', solver_options={"Threads": 8},
    run_lmps=False, mipgap=RUC_MIPGAPS[grid],
    load_shed_penalty=1e4, reserve_shortfall_penalty=1e3,
    reserve_factor=0.05, output_detail=3,
    prescient_sced_forecasts=True, ruc_prescience_hour=0,
    ruc_execution_hour=16, ruc_every_hours=24,
    ruc_horizon=48, sced_horizon=SCED_HORIZONS[grid],
    lmp_shortfall_costs=False,
    enforce_sced_shutdown_ramprate=False,
    no_startup_shutdown_curves=False,
    init_ruc_file=None, verbosity=0,
    output_max_decimals=4, create_plots=False,
    renew_costs=None, save_to_csv=False,
    last_conditions_file=None,
)
report_corr = sim_corr.simulate()
```

```
Found 149 actl wind columns.
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
Warning for adding constraints: zero or small (< 1e-13) coefficients, ignore
d
```

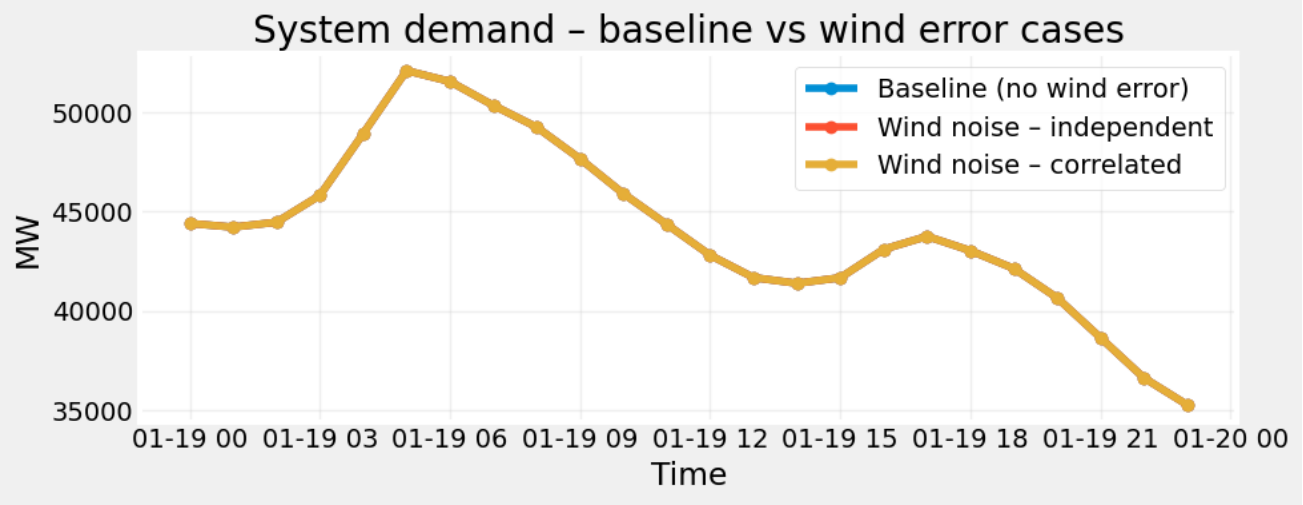
In [24]:
```
with open("report_wind_independent.pkl", "rb") as f:
```

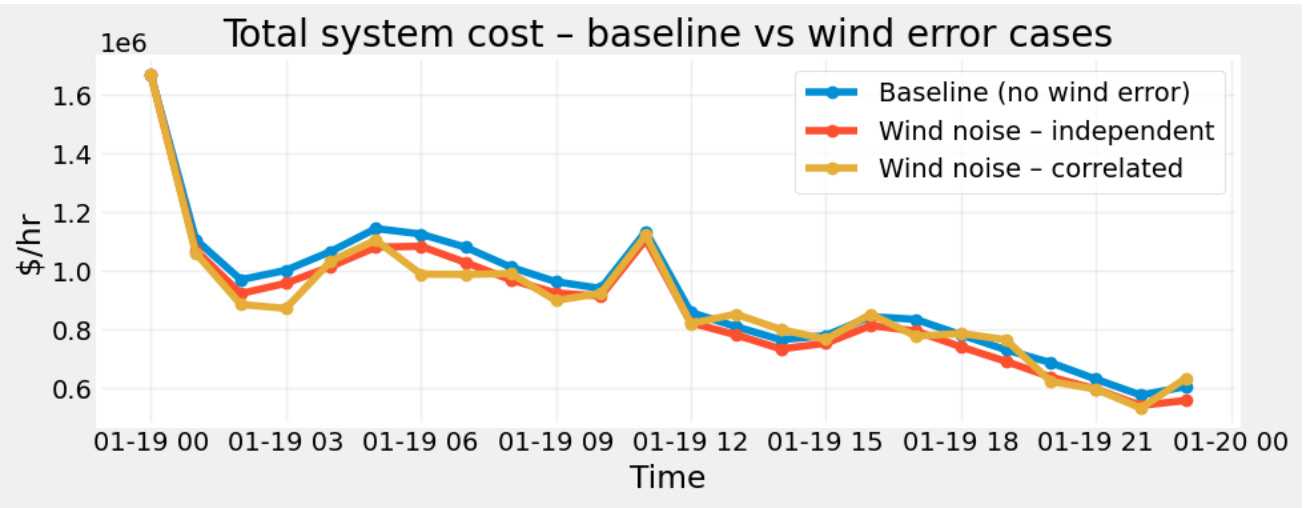
```
    report_ind = pickle.load(f)

with open("report_wind_correlated.pkl", "rb") as f:
    report_corr = pickle.load(f)
```

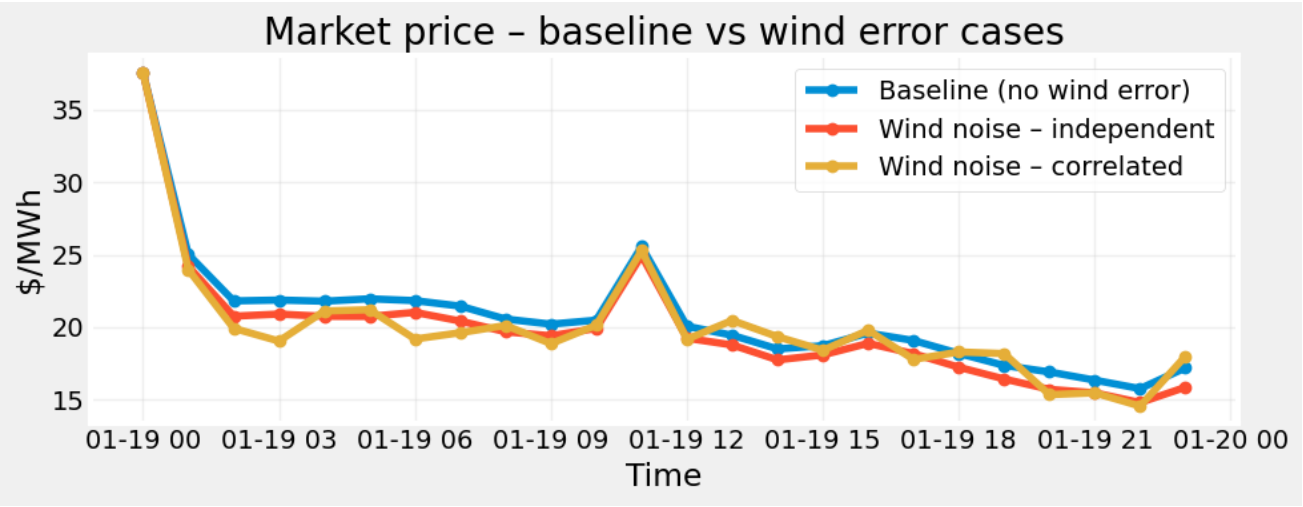
In [23]:
```
import pickle

# Save independent-noise simulation
with open("report_wind_independent.pkl", "wb") as f:
    pickle.dump(report_ind, f)

# Save correlated-noise simulation
with open("report_wind_correlated.pkl", "wb") as f:
    pickle.dump(report_corr, f)
```

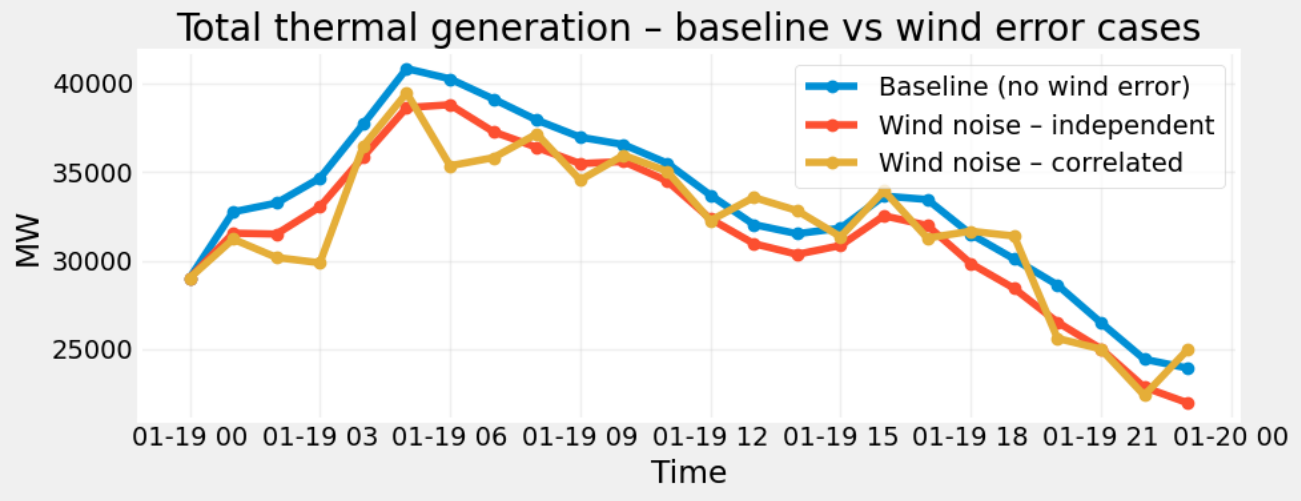
In [25]:
```
report_base = reports_by_alpha[1.0]

scenarios_raw = {
    "Baseline (no wind error)" : report_base,
    "Wind noise – independent" : report_ind,
    "Wind noise – correlated"  : report_corr,
}

def prepare_hourly(rep):
    hs = rep["hourly_summary"].copy()
    hs = hs.reset_index()

    hs["Datetime"] = pd.to_datetime(hs["Date"]) + pd.to_timedelta(hs["Hour"]

    hs["TotalCost"] = hs["FixedCosts"] + hs["VariableCosts"]

    hs["CurtailmentPct"] = (
        hs["RenewablesCurtailment"] / hs["RenewablesAvailable"]
    ) * 100.0
    hs["CurtailmentPct"] = hs["CurtailmentPct"].replace([np.inf, -np.inf], r

    hs = hs.set_index("Datetime").sort_index()
    return hs

scenario_hs = {name: prepare_hourly(rep) for name, rep in scenarios_raw.item

def plot_metric(metric, ylabel, title, scenarios=scenario_hs):
    plt.figure(figsize=(10, 4))
    for name, hs in scenarios.items():
        if metric not in hs.columns:
            continue
        plt.plot(hs.index, hs[metric], marker="o", label=name)
    plt.xlabel("Time")
    plt.ylabel(ylabel)
```

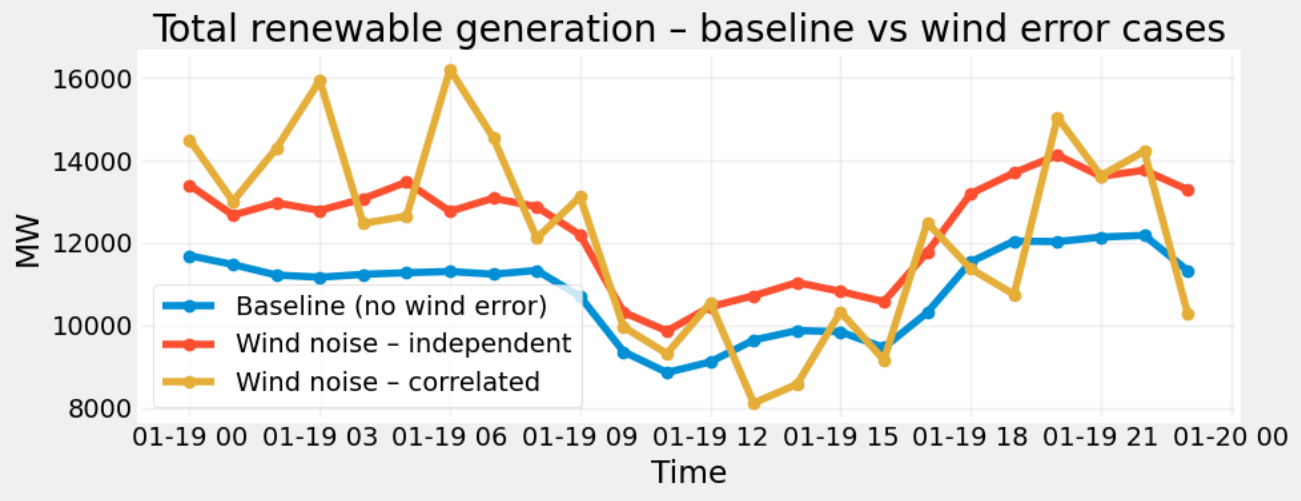
```
    plt.title(title)
    plt.grid(True, alpha=0.3)
    plt.legend()
    plt.tight_layout()

plot_metric("Demand",    "MW",    "System demand — baseline vs wind error ca
plot_metric("TotalCost", "$/hr",  "Total system cost — baseline vs wind erro
plot_metric("Price",     "$/MWh","Market price — baseline vs wind error case

def thermal_dispatch_time_series(rep):
    td = rep["thermal_detail"].copy()
    td = td.reset_index()
    td["Datetime"] = pd.to_datetime(td["Date"]) + pd.to_timedelta(td["Hour"]
    return td.groupby("Datetime")["Dispatch"].sum().sort_index()

def renewable_dispatch_time_series(rep):
    rd = rep["renew_detail"].copy()
    rd = rd.reset_index()
    rd["Datetime"] = pd.to_datetime(rd["Date"]) + pd.to_timedelta(rd["Hour"]
    return rd.groupby("Datetime")["Output"].sum().sort_index()

thermal_ts = {
    name: thermal_dispatch_time_series(rep) for name, rep in scenarios_raw.i
}
renew_ts = {
    name: renewable_dispatch_time_series(rep) for name, rep in scenarios_raw
}

plt.figure(figsize=(10, 4))
for name, series in thermal_ts.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total thermal generation — baseline vs wind error cases")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()


plt.figure(figsize=(10, 4))
for name, series in renew_ts.items():
    plt.plot(series.index, series.values, marker="o", label=name)
plt.xlabel("Time")
plt.ylabel("MW")
plt.title("Total renewable generation — baseline vs wind error cases")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
```


System demand – baseline vs wind error cases
Baseline (no wind error)
Wind noise – independent
Wind noise – correlated
MW
50000
45000
40000
35000
01-19 00
01-19 03
01-19 06
01-19 09
01-19 12
01-19 15
01-19 18
01-19 21
01-20 00
Time


Total system cost – baseline vs wind error cases
1e6
Baseline (no wind error)
Wind noise – independent
Wind noise – correlated
$/hr
1.6
1.4
1.2
1.0
0.8
0.6
01-19 00
01-19 03
01-19 06
01-19 09
01-19 12
01-19 15
01-19 18
01-19 21
01-20 00
Time


Market price – baseline vs wind error cases
Baseline (no wind error)
Wind noise – independent
Wind noise – correlated
$/MWh
35
30
25
20
15
01-19 00
01-19 03
01-19 06
01-19 09
01-19 12
01-19 15
01-19 18
01-19 21
01-20 00
Time





Comments for Part B:

We compare three cases here:

- Baseline: wind equals the original forecast (no added error).
- Independent wind errors: each wind plant gets its own iid error with mean 0.15 and volatility 0.20, so on average wind is 15% higher, but shocks are diversified across sites.
- Correlated wind errors: all plants share a strong common shock plus small local noise, so total wind still has a +15% bias on average but much larger system-level volatility.

Demand and all other inputs are kept fixed as only wind production is perturbed.

- (1) For demand curve: The three curves lie exactly on top of each other. This is expected as demand is exogenous and does not depend on how wind is realized, so introducing wind forecast errors does not change the load profile. This plot just

confirms that all differences we see later in cost, prices and dispatch come purely from the supply side (wind), not from changes in demand.

- (2) Total system cost: Total system cost in both error cases closely tracks the baseline shape over the day, but is systematically lower in most hours. This reflects the positive bias of 0.15 on average, more wind is available, so expensive thermal generation is displaced and total cost falls. The independent-error case shows a relatively smooth reduction in cost versus baseline, because positive and negative plant-level shocks partially offset each other across the fleet. The correlated-error case leads to more pronounced hour-to-hour deviations. When the common shock is high, all turbines over-produce together and system cost drops more sharply and when it is low, they under-produce together and cost moves closer to the baseline. This illustrates how shared weather shocks increase the volatility of total system cost even if the mean cost is lower.

- (3) Market Price: Market prices follow the same pattern as total system cost. In both error cases, prices are generally below the baseline, consistent with extra low-marginal-cost wind pushing more expensive thermal units out of the merit order. With independent errors, the price reduction is fairly smooth, because spatial diversification of wind smooths the net supply curve. With correlated errors, prices are slightly more volatile around the baseline path. When the common wind shock is high, prices drop more (cheap wind sets the margin more often) and when the shock is low, prices revert toward the baseline. This shows that correlated wind risks translate directly into more volatile price outcomes, even if the average price level is lower.

- (4) For thermal generation: For all hours, baseline thermal generation is the highest, and both noise cases lie below it. This reflects the basic substitution. More realized wind implies less thermal output. The independent-error curve is again relatively smooth and consistently below the baseline, meaning the system uses less thermal energy in most hours thanks to the +15% expected wind. In the correlated-error case, thermal generation fluctuates more strongly around the independent curve. So, common high-wind hours cause a sharp drop in thermal dispatch, while common low-wind hours push thermal closer to the baseline level. This highlights that correlated wind errors create larger swings in thermal dispatch, increasing operational variability for thermal plants.

- (5) For renewable generation: As expected, baseline renewables are the lowest across most hours, since there is no positive bias added. The independent-error case has a higher, but relatively smooth, renewable profile. Turbine-level errors

partially cancel, so aggregate wind is more stable even though it is higher on average. The correlated-error case exhibits both the highest average renewable generation and the largest intraday volatility. When the common shock is positive, all turbines move together and total wind spikes. When it is negative, total wind drops sharply. This demonstrates that shared weather shocks reduce geographic diversification and amplify system-level renewable volatility.

In [ ]: